# 1:SpringAMQP整合 rabbitmq

**一**:**通过**rabbitmqAdmin **来声明交换机，队列,绑定**

1.1）**需要的**jar**包**

```
<!--rabbitmq连接对象-->
 <dependency>
     <groupId>com.rabbitmq</groupId>
     <artifactId>amqp-client</artifactId>
     <version>3.6.5</version>
 </dependency>
<dependency>
 <groupId>org.springframework.amqp</groupId>
     <artifactId>spring-rabbit</artifactId>
     <version>1.7.6.RELEASE</version>
 </dependency>
```

1.2）1.2**测试**rabbitmqAdmin**接口**

```
/**
 * 配置创建连接工厂
 * @return
 */
 @Bean
 public ConnectionFactory connectionFactory() {
     CachingConnectionFactory cachingConnectionFactory = new CachingConnectionFactory();
     cachingConnectionFactory.setAddresses("47.104.128.12:5672");
     cachingConnectionFactory.setUsername("guest");
     cachingConnectionFactory.setPassword("guest");
     cachingConnectionFactory.setVirtualHost("cloudmall");
     return cachingConnectionFactory;
 }
@Bean
 public RabbitAdmin rabbitAdmin(ConnectionFactory connectionFactory) {
     RabbitAdmin rabbitAdmin = new RabbitAdmin(connectionFactory);
     rabbitAdmin.setAutoStartup(true);
     return rabbitAdmin;
 }

 @Test
        public void testRabbitAdmin() {
                //声明一个直接交换机
                rabbitAdmin.declareExchange(new DirectExchange("temp.direct",true,false,null));
                //申明一个主题交换机
                rabbitAdmin.declareExchange(new TopicExchange("temp.topic",true,false));
                //申明一个扇形交换机
                rabbitAdmin.declareExchange(new FanoutExchange("temp.fanout",true,false));

                //声明队列
                rabbitAdmin.declareQueue(new Queue("temp.queue.direct",true));

                rabbitAdmin.declareQueue(new Queue("temp.queue.topic",true));
```

```
            rabbitAdmin.declareQueue(new Queue("temp.queue.fanout",true));

            //声明bingding
            rabbitAdmin.declareBinding(new Binding("temp.queue.direct", Binding.DestinationType.QUEUE,"temp.direct","te

            rabbitAdmin.declareBinding(new Binding("temp.queue.topic", Binding.DestinationType.QUEUE,"temp.topic","tem

            rabbitAdmin.declareBinding(new Binding("temp.queue.fanout", Binding.DestinationType.QUEUE,"temp.fanout","

        }
```

1.2）**测试**RabbitTemplate

```
/**
 * rabbitmq操作模版类
 * @param connectionFactory
 * @return
 */
 @Bean
 public RabbitTemplate rabbitTemplate (ConnectionFactory connectionFactory) {
 RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
   return rabbitTemplate;
 }

 ===========================================s申明三个交换机======================

    @Bean
   public DirectExchange directExchange() {
       return new DirectExchange("directExchange001",true,false);
   }

   @Bean
   public TopicExchange topicExchange() {
       return new TopicExchange("topicExchange002",true,false);
   }

   @Bean
   public FanoutExchange fanoutExchange() {
       return new FanoutExchange("fanoutExchange003",true,false);
   }

====================================================申明三个队列================
 /**
    * 声明队列
    */
   @Bean
   public Queue queue001() {
       return new Queue("queue001",true,false,false);
   }
   @Bean
   public Queue queue002() {
       return new Queue("queue002",true,false,false);
   }

   @Bean
```

```
    public Queue queue003() {
        return new Queue("queue003",true,false,false);
    }
=====================================================确定绑定关系================
====================================直接交换机绑定了二个队列（queue02 queue03）
    /**
     * 队列二绑定到DirectExchange direct.key.key1
     * @return
     */
    @Bean
    public Binding queue002BindingDirectExchange() {
        return BindingBuilder.bind(queue002()).to(directExchange()).with("direct.key.key1");
    }

     /**
     *队列三绑定到 DirectExchange direct.key.key2
     * @return
     */
    @Bean
    public Binding queue003BindingDirectExchange() {
        return BindingBuilder.bind(queue003()).to(directExchange()).with("direct.key.key2");
    }
=================================主题交换机绑定二个队列(queue01 queue03)
    /**
     * queue001 通过 top.key.*绑定到 topicExchange
     * @return
     */
    @Bean
    public Binding queue001BindingTopicExchange002() {
        return BindingBuilder.bind(queue001()).to(topicExchange()).with("topic.key.*");
    }

        /**
     *队列三绑定到 topic.#
     * @return
     */
    @Bean
    public Binding queue003BindingTopicExchange() {
        return BindingBuilder.bind(queue003()).to(topicExchange()).with("topic.#");
    }


    ==========================扇形交换机绑定二个队列(queue01,queue02)
        /**
     * queue001绑定到 fanoutExchange
     * @return
     */
    @Bean
    public Binding queue001BindingFanoutExchange() {
        return BindingBuilder.bind(queue001()).to(fanoutExchange());
    }

        /**
     * 队列二绑定到FanoutExchange
     * @return
     */
    @Bean
    public Binding queue002BindingFanoutExchange001() {
        return BindingBuilder.bind(queue002()).to(fanoutExchange());
    }
```

1.3)**测试代码**:

```
@Test
 public void testRabbitTemplateToDirecit() {
    MessageProperties messageProperties = new MessageProperties();
    messageProperties.getHeaders().put("desc","消息描述");
    Message message = new Message("测试rabbitmqTemplate".getBytes(),messageProperties);
    rabbitTemplate.convertAndSend("directExchange001","direct.key.key1",message);
 }
@Test
 public void testRabbitTemplateToTopic() {
     MessageProperties messageProperties = new MessageProperties();
    messageProperties.getHeaders().put("desc","消息描述");
    Message message = new Message("测试rabbitmqTemplate".getBytes(),messageProperties);
    rabbitTemplate.convertAndSend("topicExchange002", "topic.key.key2", message, new MessagePostProcessor() {
    @Override
    public Message postProcessMessage(Message message) throws AmqpException {
    System.out.println("调用MessagePostProcessor处理消息");
    message.getMessageProperties().getHeaders().put("remark","消息remard");
    return message;
    }
    });
 }
@Test
 public void testRabbitTemplateToFanout() {
    rabbitTemplate.convertAndSend("fanoutExchange003","","测试fanout交换机");
 }
```

**二**:SimpleMessageListenerContriner **简单消息容器**

2.1)**作用**

1:**可以配置消费者配置项**

2:)**可以监听多个队列，自动启动，自动声明功能**

3:**设置事物相关的配置**

4)**设置消费者的数据量,批量消费**

5）**设置签收模式，是否重回队列，异常捕获。**

6）**消费者标签生成策略**

7）**设置监听器，转化器**

8）**可以支持动态修改消费者的参数配置**

2.2)代码演示

/**
* 自定义消费端的配置
* @return
*/
@Bean

```
public SimpleMessageListenerContainer simpleMessageListenerContainer() {
 SimpleMessageListenerContainer messageListenerContainer = new SimpleMessageListenerContainer(connectionFactory()
 //监听的队列
 messageListenerContainer.addQueues(queue001(),queue002(),queue003());
 //设置当前的消费者个数
 messageListenerContainer.setConcurrentConsumers(1);
 //设置最大消费者个数
 messageListenerContainer.setMaxConcurrentConsumers(5);
 //设置签收模式
 messageListenerContainer.setAcknowledgeMode(AcknowledgeMode.AUTO);
 //拒绝重回队列
 messageListenerContainer.setDefaultRequeueRejected(false);
 //消费端标签
 messageListenerContainer.setConsumerTagStrategy(new ConsumerTagStrategy() {
 @Override
 public String createConsumerTag(String queue) {
 return queue+":"+queue.hashCode()+ UUID.randomUUID().toString();
 }
 });
 //设置消费者
 messageListenerContainer.setMessageListener(new ChannelAwareMessageListener() {
@Override
 public void onMessage(Message message, Channel channel) throws Exception {
 System.out.println("消费的消息:"+new String(message.getBody()));
 }
 });
 return messageListenerContainer;
 }
```

## **三.**SimpleMessageListenerContainer **通过设置**messageAdapter**来设置消息消费者**

**代码演示**:

```
@Bean
 public SimpleMessageListenerContainer simpleMessageListenerContainerWithMessageAdapter() {
 SimpleMessageListenerContainer messageListenerContainer = new SimpleMessageListenerContainer(connectionFactory()
 //监听的队列
 messageListenerContainer.addQueues(queue001(),queue002(),queue003());
 //设置当前的消费者个数
 messageListenerContainer.setConcurrentConsumers(1);
 //设置最大消费者个数
 messageListenerContainer.setMaxConcurrentConsumers(5);
 //设置签收模式
 messageListenerContainer.setAcknowledgeMode(AcknowledgeMode.AUTO);
 //拒绝重回队列
 messageListenerContainer.setDefaultRequeueRejected(false);
 //消费端标签
 messageListenerContainer.setConsumerTagStrategy(new ConsumerTagStrategy() {
```

```
@Override
public String createConsumerTag(String queue) {
return queue+":"+queue.hashCode()+ UUID.randomUUID().toString();
}
});
//消息监听适配器
MessageListenerAdapter messageListenerAdapter = new MessageListenerAdapter(new MessageDelegate());
//指定消费消息的方法
messageListenerAdapter.setDefaultListenerMethod("consumerMsg");
//设置消息转化器
messageListenerAdapter.setMessageConverter(new TextMessageConverter());
messageListenerContainer.setMessageListener(messageListenerAdapter);
return messageListenerContainer;
}


消息消费的委托者
public class MessageDelegate {

  public void handleMessage(byte[] bodys) {
      System.out.println("消費消息handleMessage:"+new String(bodys));
  }

   public void handleMessage(String msg) {
      System.out.println("消費消息handleMessage:"+msg);
   }

/*    public void consumerMsg(byte[] bodys) {
      System.out.println("消費消息consumerMsg:"+new String(bodys));
   }*/

/*    public void consumerMsg(String msg) {
      System.out.println("消費消息consumerMsg:"+msg);
   }*/

   消息转化器
   public class TextMessageConverter implements MessageConverter {
   @Override
   public Message toMessage(Object object, MessageProperties messageProperties) throws MessageConversionException
      return new Message(object.toString().getBytes(),messageProperties);
   }

   @Override
   public Object fromMessage(Message message) throws MessageConversionException {
      if(message.getMessageProperties().getContentType().contains("text")) {
         return new String(message.getBody());
      }
      return message.getBody();
   }
}
}
```

**四**:springboot **整合**rabbitmq

4.1**）依赖包**

```
<dependency>
```

```
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-amqp</artifactId>
 </dependency>
```

4.2）生产端代码

application**配置**:

```
spring.rabbitmq.addresses=47.104.128.12:5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
spring.rabbitmq.virtual-host=/
spring.rabbitmq.connection-timeout=15000
spring.rabbitmq.publisher-confirms=true  生产端开启确认功能
spring.rabbitmq.publisher-returns=true #处理消息不可达的回调
spring.rabbitmq.template.mandatory=true关闭自动签收功能
```

**生产端代码**

```
public void sendMessage(Object msgContext,Map<String,Object> msgProps) {
 rabbitTemplate.setConfirmCallback(angleConfirmCallBack);
 rabbitTemplate.setReturnCallback(angleReturnCallBack);
MessageHeaders messageHeaders = new MessageHeaders(msgProps);
 Message message = MessageBuilder.createMessage(msgContext,messageHeaders);
 String msgId = UUID.randomUUID().toString();
 System.out.println("生成的全局唯一性ID"+msgId);
 CorrelationData correlationData = new CorrelationData(msgId);
//rabbitTemplate.convertAndSend("exchange-1","springboot.test",message,correlationData);
 rabbitTemplate.convertAndSend("order.exchange","order.test",message,correlationData);
 }


消息确认回调
public class AngleConfirmCallBack implements RabbitTemplate.ConfirmCallback {
    @Override
    public void  confirm(CorrelationData correlationData, boolean ack, String cause) {
        System.out.println("消息确认....................");
        System.out.println("消息唯一ID:"+correlationData.getId());
        System.out.println("消息是否签收:"+ack);
        System.out.println("消息错误原因:"+cause);
        if(!ack) {
            System.out.println("做消息可靠性投递");
        }
    }
}

消息不可达回调
public class AngleReturnCallBack implements RabbitTemplate.ReturnCallback {
    @Override
    public void returnedMessage(Message message, int replyCode, String replyText, String exchange, String routingKey) {
        System.out.println("message:"+message);
        System.out.println("replyCode:"+replyCode);
        System.out.println("replyText:"+replyText);
        System.out.println("exchange:"+exchange);
        System.out.println("routingKey:"+routingKey);

    }
}
```

4.3）消费端代码配置

```
spring.rabbitmq.addresses=47.104.128.12:5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
spring.rabbitmq.virtual-host=/
spring.rabbitmq.connection-timeout=15000
spring.rabbitmq.listener.simple.acknowledge-mode=manual
spring.rabbitmq.listener.simple.concurrency=5
spring.rabbitmq.listener.simple.max-concurrency=10
server.port=8081
```

**消费端消费配置**1()

```
@Bean
 public Queue orderQueue() {
 return new Queue("order.queue");
 }
 @Bean
 public Queue orderQueue2() {
 return new Queue("order.queue2");
 }
 @Bean
 public TopicExchange topicExchange() {
 return new TopicExchange("order.exchange");
 }

@Bean
 public Binding binding() {
 return BindingBuilder.bind(orderQueue()).to(topicExchange()).with("order.#");
 }
@Bean
 public Binding binding2() {
 return BindingBuilder.bind(orderQueue2()).to(topicExchange()).with("order.*");
 }
}


    @RabbitListener(queues = "order.queue")
    public void msgConsumer(Message message, Channel channel) throws IOException {
        System.out.println("消费消息:"+message.getPayload());
        Long deliveryTag = (Long) message.getHeaders().get(AmqpHeaders.DELIVERY_TAG);
        channel.basicAck(deliveryTag,false);
    }

    @RabbitListener(queues = "order.queue2")
    public void msgConsumer2(@Payload Order order, Channel channel, @Headers  Map<String,Object> headers) throws I
        System.out.println("消费消息:"+order);
        Long deliveryTag = (Long) headers.get(AmqpHeaders.DELIVERY_TAG);
        channel.basicAck(deliveryTag,false);
    }
}
```

**消费端消费配置2()在RabbitListener注解进行队列申明绑定**

```
@Component
public class RabbitReceiver {

 @RabbitListener(bindings = @QueueBinding(
 value = @Queue(value = "queue-1",
 durable="true"),
 exchange = @Exchange(value = "exchange-1",
 durable="true",
 type= "topic",
 ignoreDeclarationExceptions = "true"),
 key = "springboot.*"
 )
 )
 @RabbitHandler
 public void onMessage(Message message, Channel channel) throws Exception {
 System.err.println("--------------------------------------");
 System.err.println("消费端Payload: " + message.getPayload());
 Long deliveryTag = (Long)message.getHeaders().get(AmqpHeaders.DELIVERY_TAG);
 System.out.println("消費端消息:"+message);
 //手工ACK
 channel.basicAck(deliveryTag, false);
 }



 /**
 *
 * spring.rabbitmq.listener.order.queue.name=queue-2
 spring.rabbitmq.listener.order.queue.durable=true
 spring.rabbitmq.listener.order.exchange.name=exchange-1
 spring.rabbitmq.listener.order.exchange.durable=true
 spring.rabbitmq.listener.order.exchange.type=topic
 spring.rabbitmq.listener.order.exchange.ignoreDeclarationExceptions=true
 spring.rabbitmq.listener.order.key=springboot.*
 * @param order
 * @param channel
 * @param headers
 * @throws Exception
 */
 @RabbitListener(bindings = @QueueBinding(
 value = @Queue(value = "${spring.rabbitmq.listener.order.queue.name}",
 durable="${spring.rabbitmq.listener.order.queue.durable}"),
 exchange = @Exchange(value = "${spring.rabbitmq.listener.order.exchange.name}",
 durable="${spring.rabbitmq.listener.order.exchange.durable}",
 type= "${spring.rabbitmq.listener.order.exchange.type}",
 ignoreDeclarationExceptions = "${spring.rabbitmq.listener.order.exchange.ignoreDeclarationExceptions}"),
 key = "${spring.rabbitmq.listener.order.key}"
 )
 )
 @RabbitHandler
 public void onOrderMessage(@Payload com.bfxy.springboot.entity.Order order,
 Channel channel,
 @Headers Map<String, Object> headers) throws Exception {
 System.err.println("--------------------------------------");
 System.err.println("消费端order: " + order.getId());
 Long deliveryTag = (Long)headers.get(AmqpHeaders.DELIVERY_TAG);
 //手工ACK
 channel.basicAck(deliveryTag, false);
```

```
    }


  }
```